An Operational Semantics in Isabelle/HOL-CSP

Benoît Ballenghien
University Paris-Saclay,
Laboratoire Méthodes Formelles, France

Burkhart Wolff[1]
University Paris-Saclay,
Laboratoire Méthodes Formelles, France

Abstract

The theory of Communicating Sequential Processes going back to Hoare and Roscoe is still today a reference model for concurrency. In the fairly rich literature, several versions of operational semantics have been discussed, which should be consistent with the denotational one.

This work is based on Isabelle/HOL-CSP 2.0, a shallow embedding of the failure-divergence model of denotational semantics proposed by Hoare, Roscoe and Brookes in the eighties. In several ways, HOL-CSP is actually an extension of the original setting in the sense that it admits higher-order processes and infinite alphabets.

In this paper, we present a construction and formal equivalence proofs between operational CSP semantics and the underlying denotational failure-divergence semantics. The construction is based on a *definition* of the operational transition operator $P \leadsto_e P'$ basically via the *After* operator and the classical failure-divergence refinement.

Several choices are discussed to formally derive the operational semantics leading to subtle differences. The derived operational semantics for symbolic Labelled Transition Systems (LTSs) can be potentially used for certifications of model-checker logs as well as combined proof techniques.

2012 ACM Subject Classification Theory of computation → Higher order logic; Theory of computation → Semantics and reasoning; General and reference → Verification

Keywords and phrases Process-Algebra, Semantics, Concurrency, Computational Models, Theorem Proving, Isabelle/HOL

Digital Object Identifier 10.4230/LIPIcs.ITP.2024.18

Funding *Benoît Ballenghien*: Labex DigiCosme

Acknowledgements We would like to thank Catherine Dubois for her remarks on earliest versions of this paper. The present version has been generated and deeply checked with Isabelle/DOF [8, 9].

1 Introduction

Communicating Sequential Processes (CSP) is a language to specify and verify patterns of interaction of concurrent systems. Together with CCS and LOTOS, it belongs to the family of *process algebras*. CSP's rich theory comprises denotational, operational and algebraic semantic facets and has influenced programming languages such as Limbo, Crystal, Clojure and most notably Golang [14]. CSP has been applied in industry as a tool for specifying and verifying the concurrent aspects of hardware systems, such as the T9000 transputer [5].

The theory of CSP was first described in 1978 by Tony Hoare [15], but has since evolved substantially [6, 7, 27]. The denotational semantics of CSP is described by a fully abstract model of behaviour designed to be *compositional*: a process P encompasses all possible behaviours, i. e. sets of *traces* annotated by additional information that allow to reason over deadlocks (the resulting semantic domain is called *failure semantics* F)

[1] corresponding author

- and additionally livelocks (the *failure-divergence semantics FD*).

Several attempts have been undertaken to formalize this fairly complex theory, notably [10, 16, 19, 23, 34]. The arguably (see Section 7) most comprehensive one is HOL-CSP [3, 4, 31, 33, 34], which is in several ways not only a formalization in a proof assistant, but a generalization of the original setting:

- the set of traces *'α trace* is constructed over an arbitrary type *'α* in HOL, paving the way for dense time, vector spaces, etc, [2],
- more generally speaking, HOL-CSP attempts to remove finiteness-restrictions, and
- the semantic domain is encapsulated in the type *'α process* belonging to the cpo type class (see Subsection 2.4). Thus, process patterns can be expressed and analyzed.

In this paper, we present the formal theory of the operational semantics of HOL-CSP which is consistent with the denotational one by construction. To this end, we proceed by defining the operational transition operator $P \leadsto_e P'$ in terms of the *After* operator and the classical failure-divergence refinement. In the literature on process algebras like CSP ([6, 7, 15, 27]) or Circus ([35, 36]), several versions of operational semantics have been presented, but a formal proof of equivalence for a substantial part of the language has never been undertaken. Our proof architecture foresees an Isabelle locale (a parameterized theory) whose instances represent several of these versions, thus shedding some light on their relationships. The operational rules for small-step and big-steps pave the way for symbolic execution of processes and the combination of model-checking with theorem proving.

We proceed as follows: after an introduction to "classic" CSP and our extensions HOL-CSP and HOL-CSPM as an Isabelle framework in Section 2, we present in Section 3 the core construction of this paper, the *bridge*-definition linking the denotational semantics to operational one(s), resulting in the formally proven laws written in Section 4. This is generalized to big steps semantics in Section 5 and possible variants are discussed in Section 6. Note that HOL-CSP[33], HOL-CSPM[3] as well as the novel contribution HOL-CSP_OpSem [4] containing the proofs discussed here are published in the Archive of Formal Proofs AFP[3].

2 Background

2.1 Classic CSP Syntax

At a glance, the syntax of the classical CSP core language reads as follows:

$$P ::= SKIP \mid STOP \mid P \,\square\, P' \mid P \sqcap P' \mid P \,[\![A]\!]\, P' \mid P \,;\, P' \mid P \setminus A$$
$$\mid a \to P \mid \square a \in A \to P(a) \mid \sqcap a \in A \to P(a) \mid Renaming\ P\ g \mid \mu\ X.\ f(X)$$

SKIP signals termination, *STOP* denotes a deadlock. Two choice operators are distinguished:

1. the *external choice* -□-, which forces a process "to follow" whatever its context requires,
2. the *internal choice* -⊓-, which imposes on the context of a process "to follow" the non-deterministic choices made.

With the prefix operator $a \to P$ which signals a and continues with P (where a is an element of a set Σ of *events*), generalized choices of the form $\square a{\in}A \to P(a)$ resp. $\sqcap a{\in}A \to P(a)$ are constructed (A is originally a finite set). When events are tagged with *channels*, i. e. $\Sigma = CHANNELS \times DATA$, syntactic sugar like $c?x{\in}A{\to}P(x)$ or $c!x{\in}A{\to}P(x)$ is added; the

[2] or even differential equations as in cyber-physical system models [12]
[3] For the developer version, see https://gitlab.lisn.upsaclay.fr/burkhart.wolff/hol-csp2.0/.

former reads intuitively as "x is read from channel c" while the latter means "x is arbitrarily chosen from A and sent into c" (where $c \in CHANNELS$ and $x \in DATA$).

The sequential composition $P\ ;\ P'$ behaves first like P and, once it has successfully terminated, like P'. $P \setminus A$ consists in hiding the events of the set A. *Renaming* $P\ g$ results in a process in which each event e of P is renamed in $g(e)$. The *Sliding* operator $P \rhd P'$ is defined as $(P \Box P') \sqcap P'$. The fixed point $\mu\ X.\ f(X)$ operator satisfies $(\mu\ X.\ f(X)) = f\ (\mu\ X.\ f(X))$ (but requires precautions, see Subsection 2.4).

CSP describes all communication with one single primitive: the synchronized product written $P\ [\![A]\!]\ P'$. Note that interleaving $P\ |||\ P'$ stands for $P\ [\![\{\}]\!]\ P'$, whereas the parallel operator $P\ ||\ P'$ is a shortcut for $P\ [\![UNIV]\!]\ P'$ ($UNIV$ is the universal set).

2.2 Classic CSP Semantics

The denotational semantics (following [27]) comes in three layers: the *trace model*, the *(stable) failures model* and the *failure-divergence model*.

In the trace semantics model, the behaviour of a process P is denoted by a prefix-closed set of traces, denoted $\mathcal{T}\ P$, similar to the well-known concept of a "language of an automata". Since traces are finite lists and infinite behaviour is therefore represented via the set of approximations, an additional element *tick* (written ✓) is used to represent explicit termination signalized by *SKIP*. Note that, obviously, *tick* should only appear at the end of a trace (traces should be *front-tickFree*).

It is impossible to distinguish external and internal non-determinism in the trace model since the traces of both operators are just the union of their argument traces. To be more discriminant, [6] proposed the failure semantics model, where traces were annotated with a set of *refusals*, i. e. sets of events a process can *not* engage in. This leads to the notion of a *failure* $(t,\ X) \in \mathcal{F}\ P$ which is a pair of a trace t and a set of refusals X. Consider for example the process $P = (a \rightarrow SKIP) \Box (a \rightarrow STOP)$. The traces $\mathcal{T}\ P$ will non-deterministically lead to a situation where the process accepts termination (but refuses everything else) or just refuses everything. So, if we assume $\Sigma = \{a, ✓\}$, then the traces $\mathcal{T}\ P$ will be $\{[], [a]\}$. The failures $\mathcal{F}\ P$ are then $\{([], \{\{✓\}\}), ([a], \{\Sigma, \{a\}\})\}$ (plus all subsets of the respective refusal sets, which is required for the recursion ordering discussed in Subsection 2.4).

Finally, [6] enriched the semantic domain of CSP with one more element, the set of *divergences* (written $\mathcal{D}\ P$), in order to distinguish deadlocks from livelocks[4]. In the failure divergence model, the semantic domain consists of a pair of failures and divergences, where the latter are traces to situations where livelocks may occur.

In contrast to Hoare Logics and its Hoare Triples, which is a framework to reason over terminating calculations, CSP and process refinement are designed to reason over non-terminating calculations. Three classic refinement notions are considered:

1. the trace refinement: $P \sqsubseteq_T Q \equiv \mathcal{T}\ P \supseteq \mathcal{T}\ Q$,
2. the failure refinement: $P \sqsubseteq_F Q \equiv \mathcal{F}\ P \supseteq \mathcal{F}\ Q$, and
3. the failure-divergence refinement $P \sqsubseteq_{FD} Q \equiv \mathcal{F}\ P \supseteq \mathcal{F}\ Q \wedge \mathcal{D}\ P \supseteq \mathcal{D}\ Q$.

It turns out that beyond common protocol refinement proofs and test-problems, many properties such as deadlock or livelock freeness can be expressed via a refinement statement.

[4] also called *infinite internal chatter* as occurring in processes like $\mu\ x.\ a \rightarrow x \setminus \{a\}$

2.3 Theories and Locales in Isabelle and HOL

Isabelle is a major interactive proof assistant implementing higher-order logic (HOL). As an LCF style theorem prover, it is based on a small logical core (kernel) to increase the trustworthiness of proofs without requiring — yet supporting — explicit proof objects.

The Isabelle distribution comes with a number of library theories constructed solely from definitional axioms; among them basic data-types for sets, lists, arithmetics, a substantial part of analysis, and — particularly relevant here — Scott domain theory (`HOLCF`) [22] providing a particular type class $'\alpha$::*pcpo*, i.e. the class of types $'\alpha$ for which a least element $\bot$ and a complete partial order -$\sqsubseteq$- is defined.

`HOLCF` provides the concept of *continuity*, the concept of *admissibility*, the fixed point operator $\mu\ x.\ f\ x$ as well as the fixed point induction for admissible predicates. Isabelle's type inference can automatically infer, for example, that if $'\alpha$::*pcpo*, then $('\beta \Rightarrow '\alpha)$::*pcpo*.

A distinguishing feature of Isabelle is the **locale** mechanism, i. e. theories that may be parameterized by types, constant-symbols and local hypotheses over them. Since **locale**s may inherit from other **locale**s, they represent a powerful structuring mechanism for orders and algebraic structures very similar to dependent types available in other systems.

2.4 Isabelle/HOL-CSP

`Isabelle/HOL-CSP` is a shallow embedding of CSP in HOL based on the traditional semantic domain described by 9 "axioms" over the three semantic functions $\mathcal{T} :: '\alpha\ process \Rightarrow '\alpha\ trace\ set$, $\mathcal{F} :: '\alpha\ process \Rightarrow '\alpha\ failure\ set$ and $\mathcal{D} :: '\alpha\ process \Rightarrow '\alpha\ trace\ set$:

- the empty trace is always the initial trace and any trace is *front-tickFree*;
- traces of a process are *prefix-closed* and a process can refuse all subsets of refusals;
- any event refused by a process after a trace s must be in a refusal set associated to s;
- the *tick* accepted after a trace s implies that all other events are refused;
- a divergence trace with any suffix is itself a divergence one
- once a process has diverged, it can engage in or refuse any sequence of events.
- a *tick*-ending divergence trace has a *tickFree* divergence trace prefix of maximal length.

More formally, a process P of the type Σ *process* should have the following properties:

$$
\begin{aligned}
&([] \in \mathcal{T}\ P \wedge (\forall\, s\ X.\ (s,\ X) \in \mathcal{F}\ P \longrightarrow \mathit{front\text{-}tickFree}\ s) \wedge\\
&(\forall\, s\ t.\ s\ @\ t \in \mathcal{T}\ P \longrightarrow s \in \mathcal{T}\ P) \wedge (\forall\, s\ X\ Y.\ (s,\ Y) \in \mathcal{F}\ P \wedge X \subseteq Y \longrightarrow (s,\ X) \in \mathcal{F}\ P) \wedge\\
&(\forall\, s\ X\ Y.\ (s,\ X) \in \mathcal{F}\ P \wedge (\forall\, c.\ c \in Y \longrightarrow s\ @\ [c] \notin \mathcal{T}\ P) \longrightarrow (s,\ X \cup Y) \in \mathcal{F}\ P) \wedge\\
&(\forall\, s\ X.\ s\ @\ [\checkmark] \in \mathcal{T}\ P \longrightarrow (s,\ X - \{\checkmark\}) \in \mathcal{F}\ P) \wedge\\
&(\forall\, s\ t.\ s \in \mathcal{D}\ P \wedge \mathit{tickFree}\ s \wedge \mathit{front\text{-}tickFree}\ t \longrightarrow s\ @\ t \in \mathcal{D}\ P) \wedge\\
&(\forall\, s\ X.\ s \in \mathcal{D}\ P \longrightarrow (s,\ X) \in \mathcal{F}\ P) \wedge (\forall\, s.\ s\ @\ [\checkmark] \in \mathcal{D}\ P \longrightarrow s \in \mathcal{D}\ P))
\end{aligned}
$$

The core of `HOL-CSP` is to encapsulate this wishlist into a type definition. This is achieved by 1) defining the pair of failures and divergences $\Sigma\ process_0$ via $(\Sigma^{\checkmark}\ list \times \Sigma^{\checkmark}\ set)\ set \times (\Sigma^{\checkmark}\ list)\ set$ (where $\Sigma^{\checkmark} = \Sigma \uplus \{\checkmark\}$), 2) by turning the above wishlist into a data-constraint *is-process* of type $\Sigma\ process_0 \Rightarrow bool$, and 3) by establishing an isomorphism between the subset of $\Sigma\ process_0$'es satisfying *is-process* via the-specification construct:

$$\textbf{typedef}\ '\alpha\ process = \{P :: '\alpha\ process_0\ .\ is\text{-}process\ P\}$$

Subsequently, we provide definitions for each CSP operator in terms of $\Sigma\ process_0$; these definitions formalize directly the textbook [27]. Finally, we prove that each operator preserves the *is-process*-invariant. The preservation even holds for arbitrary (possibly infinite) sets A in the generalisations $\Box x{\in}A \rightarrow P(x)$ resp. $\sqcap x{\in}A \rightarrow P(x)$. Note that both use higher-order abstract syntax and have the type $'\alpha\ set \Rightarrow ('\alpha \Rightarrow '\alpha\ process) \Rightarrow '\alpha\ process$.

A major problem prevails: how to give semantics to the fixed point operator?
This is achieved by turning the denotational domain of CSP into a Scott complete partial order (cpo) [29], which provides semantics for the fixed point operator $\mu\ x.\ f(x)$ under the condition that f is continuous wrt. this partial ordering. Since the natural ordering - $\sqsubseteq_{FD}$ - is too weak for this purpose, Roscoe and Brookes [24] proposed a complete *process ordering* $P \sqsubseteq Q$ which is stronger, i. e. $P \sqsubseteq Q \Longrightarrow P \sqsubseteq_{FD} Q$, and yet ensures completeness at least for general read and write operations.
It is based on the concept *refusals after* a trace s ($\mathcal{R}_a\ P\ s \equiv \{X \mid (s, X) \in \mathcal{F}\ P\}$):

$$P \sqsubseteq Q \equiv \mathcal{D}\ Q \subseteq \mathcal{D}\ P \wedge (\forall\, s.\ s \notin \mathcal{D}\ P \longrightarrow \mathcal{R}_a\ P\ s = \mathcal{R}_a\ Q\ s) \wedge \mathit{min\text{-}elems}\ (\mathcal{D}\ P) \subseteq \mathcal{T}\ Q$$

▶ **Theorem 1** (Continuity). *Almost all HOL-CSP operators $\otimes$ are continuous wrt. $(\sqsubseteq)$, i. e.:*

$$\mathit{cont}\ f \Longrightarrow \mathit{cont}\ g \Longrightarrow \mathit{cont}(\lambda x.\ (f\ x) \otimes (g\ x))$$

▶ **Theorem 2** (Fixed-point Inductions). *Since $(\sqsubseteq_{FD})$ is admissible, when f is continuous we have an induction rule of the following form:*

$$C\ (\bot) \sqsubseteq_{FD} Q \Longrightarrow (\bigwedge x.\ C\ (x) \sqsubseteq_{FD} Q \Longrightarrow C(f\ x) \sqsubseteq_{FD} Q) \Longrightarrow C\ (\mu\ X.\ f\ X) \sqsubseteq_{FD} Q$$

▶ **Proposition 1** (CSP-Algebra). *HOL-CSP provides about 200 rules derived from the denotational semantics, be it monotonicities or equalities, which were called the "axioms" in the literature. We show here only the small collection used in the subsequent example proof:*

$$\begin{aligned}
&(\forall\, y.\ c\ y \in S) \Longrightarrow c?x \to P\ x\ [\![S]\!]\ c?x \to Q\ x = c?x \to (P\ x\ [\![S]\!]\ Q\ x)\\
&(\forall\, y.\ c\ y \in S) \Longrightarrow \mathit{inj}\ c \Longrightarrow c!a \to P\ [\![S]\!]\ c?x \to Q\ x = c!a \to (P\ [\![S]\!]\ Q\ a)\\
&d\ a \notin S \Longrightarrow (\bigwedge y.\ c\ y \in S) \Longrightarrow d!a \to P\ [\![S]\!]\ c?x \to Q\ x = d!a \to (P\ [\![S]\!]\ c?x \to Q\ x)\\
&d \in S \Longrightarrow (\bigwedge y.\ c\ y \notin S) \Longrightarrow d \to P\ [\![S]\!]\ c?x \to Q\ x = c?x \to (d \to P\ [\![S]\!]\ Q\ x)\\
&d\ a \notin C \Longrightarrow c \in C \Longrightarrow c \to Q\ [\![C]\!]\ d!a \to P = d!a \to (c \to Q\ [\![C]\!]\ P)\\
&\forall\, y.\ c\ y \notin B \Longrightarrow\ c?x \to P\ x \setminus B = c?x \to (P\ x \setminus B)\\
&\forall\, y.\ c\ y \notin B \Longrightarrow c!a \to P \setminus B = c!a \to (P \setminus B)\\
&c\ a \in B \Longrightarrow c!a \to P \setminus B = P \setminus B \qquad \textit{etc.}
\end{aligned}$$

The theories `HOL-CSP` and `HOL-CSPM` [3] also add a number of extensions of the original language. This includes for the binary operators $P\ [\![A]\!]\ P'$, $P\ ;\ P'$, $P \Box P'$, $P \sqcap P'$, the generalizations $[\![S]\!]i \in\# M.\ P(i)$, $|||\ i \in\# M.\ P(i)$, etc. Roscoe's operators *Interrupt* $P \triangle P'$ and *Throw* (exception handler) $P\ \Theta\ a \in A.\ P'(a)$ have also been included since they come in handy in some of the more general constructions. Finally, [31] proposed another refinement ordering, the trace-divergence ordering $P \sqsubseteq_{DT} Q \equiv\ P \sqsubseteq_T Q \wedge \mathcal{D}\ Q \subseteq \mathcal{D}\ P$, which has surprisingly benign properties wrt. operational semantics and which is relevant for applications [12].

2.5 A Model and Sample Proof in HOL-CSP

Of course, proving refinements is not done by unfolding the definitions in the denotational semantics. Instead, the predominant proof technique is merely fixed point induction via Theorem 2, application of the algebraic rules of Proposition 1 as well as the monotonicity rules which are a consequence of Theorem 1. We demonstrate this with the paradigmatic `CopyBuffer` example, where we model a protocol *COPY* ("received data on channel *left* will eventually be copied into channel *right*") and an implementing *SYS* which transfers the data from some *SEND*-component into some *REC*-component using an internal channel *mid* where *REC* acknowledges each data-package via a signal on the internal *ack*-channel.

The formalisation of these model-elements proceeds as follows. The events were defined by the inductive data-type introducing the channels:

datatype $'\alpha$ *channel* = *left* $'\alpha$ | *right* $'\alpha$ | *mid* $'\alpha$ | *ack*

Note that this definition leaves open what data is actually transmitted. A synchronisation set *SYN* is defined via $\{e \mid \exists\, x.\ e = mid\ x\} \cup \{ack\}$. The process *COPY* of type $'\alpha$ *channel process* is defined by $\mu\ x.\ left?xa \rightarrow (right!xa \rightarrow x)$, the process *SEND* by $\mu\ x.\ left?xa \rightarrow (mid!xa \rightarrow (ack \rightarrow x))$ and the process *REC* by $\mu\ x.\ mid?xa \rightarrow (right!xa \rightarrow (ack \rightarrow x))$. The latter two are wired together to the process *SYS* via $SYS \equiv SEND\ [\![SYN]\!]\ REC \setminus SYN$.

Now we ask the question: *does SYS implement the protocol COPY*? This can be rewritten as the following refinement problem : $COPY \sqsubseteq_{FD} SYS$.
Unfolding *COPY* and applying Theorem 2 yields the two subgoals:

1. $\bot \sqsubseteq_{FD} (SEND\ [\![SYN]\!]\ REC \setminus SYN)$
2. $\bigwedge x.\ x \sqsubseteq_{FD} (SEND\ [\![SYN]\!]\ REC \setminus SYN) \Longrightarrow$
   $left?a \rightarrow (right!a \rightarrow x) \sqsubseteq_{FD} (SEND\ [\![SYN]\!]\ REC \setminus SYN)$

where the former is trivial and the latter represents the induction step. If we unfold once *SEND* and *REC* and apply the reduction rules of Proposition 1, this results in:

$$left?a \rightarrow (right!a \rightarrow x) \sqsubseteq_{FD} left?a \rightarrow (right!a \rightarrow (SEND\ [\![SYN]\!]\ REC \setminus SYN))$$

Applying the monotonicity rules resulting from Theorem 1 we can reduce this goal to the induction hypothesis $x \sqsubseteq_{FD} (SEND\ [\![SYN]\!]\ REC \setminus SYN)$.

Furthermore this proof can be highly automated (reduces to a few lines in Isabelle/Isar). No assumption is made over $'\alpha$, this construction is therefore truly parametric over data, which is in stark contrast to model-checkers for CSP such as [1, 30]. Using the fact that functions over processes are continuous, we can specify and analyse, e. g., global variables by $VAR\ \sigma_{init} \equiv (\mu\ x.\ (\lambda\sigma.\ (Read!\sigma \rightarrow x\ \sigma) \Box (Update?\sigma' \rightarrow x\ \sigma')))\ \sigma_{init}$ and other building blocks of concurrent programs like buffers, semaphores and monitors.

3 Small Steps Semantics

Operational semantics of CSP involve two kinds of transitions that we need to define:

- the τ transition, denoted $P \leadsto_{\tau} Q$, (*internal transition*)
- and the transition with an observable event (*external transition*) where we distinguish the two cases resulting from the type sum of "real" events *ev e* and the special event ✓. The former transition will be denoted by $P \leadsto_{e} Q$, the latter by $P \leadsto_{\checkmark} Q$.

Initially, Hoare and Jifeng in [20] proposed the following link between the operational and denotational semantics: $P \leadsto_{\tau} Q \equiv P \sqsubseteq_{FD} Q$ and $P \leadsto_{e} Q \equiv P \sqsubseteq_{FD} (e \rightarrow Q) \Box P$. This approach is fine as long as we do not consider explicit termination of processes via the special event ✓. Moreover, the initial presentation was referring to fragment of the language which foundation wrt. the underlying denotational semantics (including process ordering) was still prone to subtle errors [34]. For these reasons, we opted for another bridge based on the *After* operator, denoted *P after e*, which represents a kind of inversion of $e \rightarrow P$. We will investigate the precise connection between both definitions in Section 6.

3.1 The Notion of initials

As prerequisite, we need the events P can start with, called *initials* P and denoted P^0.

▶ **Definition 1** (*initials*). *The definition is straightforward:* $P^0 \equiv \{e \mid [e] \in \mathcal{T}\ P\}$ *or equivalently, thanks to* $'\alpha$ *process properties,* $P^0 = \{e \mid \exists\, s.\ e \cdot s \in \mathcal{T}\ P\}$.

Intuitively, for each *ev e* in P^0, the traces of *P after e* should be the tails of the traces of *P* beginning with *ev e*. The question arises what happens with processes *P* where no trace is beginning with *ev e*. The semantic domain of *P after e* will require that the set of traces is non-empty (recall the *is-process* invariant from which we built the *'α process* type).

The *initials* are a notion commonly evoked in [6, 27, 28]; for now let us derive the general computational rules for it.

▶ **Theorem 3** (Basic rules for *initials*). *We derive:*

$$\begin{array}{ccc} \bot^0 = UNIV & (P^0 = \emptyset) = (P = STOP) & SKIP^0 = \{\checkmark\} \\ (P \sqcap Q)^0 = P^0 \cup Q^0 & (P \,\Box\, Q)^0 = P^0 \cup Q^0 & (e \rightarrow P)^0 = \{ev\ e\} \\ (P \rhd Q)^0 = P^0 \cup Q^0 & (P \,\triangle\, Q)^0 = P^0 \cup Q^0 & (P\ \Theta\ a{\in}A.\ Q\ a)^0 = P^0 \end{array}$$

▶ **Theorem 4** (More complex rules for *initials*). *The following list requires more caution:*

- $(Renaming\ P\ f)^0 = (\mathit{if}\ P = \bot\ \mathit{then}\ UNIV\ \mathit{else}\ EvExt\ f\ `\ P^0)$
- $(P\ ;\ Q)^0 = (\mathit{if}\ P = \bot\ \mathit{then}\ UNIV\ \mathit{else}\ P^0 - \{\checkmark\} \cup (\mathit{if}\ \checkmark \in P^0\ \mathit{then}\ Q^0\ \mathit{else}\ \emptyset))$
- $(P\ [\![S]\!]\ Q)^0 = P^0 \cup Q^0 - (\{\checkmark\} \cup ev\ `\ S) \cup P^0 \cap Q^0 \cap (\{\checkmark\} \cup ev\ `\ S)$
  if $P \neq \bot$ *and* $Q \neq \bot$ *(otherwise* $(\bot\ [\![S]\!]\ Q)^0 = UNIV$ *and* $(P\ [\![S]\!]\ \bot)^0 = UNIV$*).*

The equality for the *Hiding* operator, proved but omitted here, is downright difficult. Note that the function *initials* is of type *'α process* ⇒ *'α event set*. This implies that we may have $\checkmark \in P^0$, especially when $P = SKIP$ for which it serves as a refinement characterization.

▶ **Theorem 5** (Characterization of initial ✓). $\checkmark \in P^0$ *if and only if* $P \sqsubseteq_{FD} SKIP$.

3.2 The After Operator

There is no comprehensive treatment of the *After* operator in the CSP literature, at least not with a formal definition and a precise clarification of the behaviour wrt. the other operators; we had to do a number of trials and second-guessing here. A key element is the notion of *initials* (Definition 1); assuming $ev\ e \in P^0$ for $P{::}'\alpha\ process$, we obviously choose its failures to be $\{(s, X) \mid (ev\ e \cdot s, X) \in \mathcal{F}\ P\}$ and its divergences $\{s \mid ev\ e \cdot s \in \mathcal{D}\ P\}$.

This solves part of the problem, but is not enough. We will need a *total* definition of this operator in HOL, i. e. we need to deal with the case $ev\ e \notin P^0$. There are basically the alternatives *STOP*, ⊥, or just some underspecified constant *undefined*. Since the actual choice made leads to subtle differences in corner cases but does not impact the operational rules that we establish, we use Isabelle **locale**s mentioned in Subsection 2.3 to model this:

▶ **Definition 2** (*After* operator).

```
locale After =  fixes Ψ :: ‹['α process, 'α] ⇒ 'α process› begin

lift-definition After :: ‹['α process, 'α] ⇒ 'α process› (infixl ‹after› 77)
  is ‹λP e.   if ev e ∈ initials P
              then ({(s, X). (ev e # s, X) ∈ F P}, {s. ev e # s ∈ D P})
              else (F (Ψ P e), D (Ψ P e))›
```

Here, **lift-definition** is a variant of Isabelle constant **definition** which gives automatic support for "lifting" an operation of the $'\alpha\ process_0$-level to $'\alpha\ process$. Note that Ψ is a parameter of the locale requiring no assumption which can be instantiated freely.

The need of the theory of the *After* operator and its straightforward generalisation to traces denoted by $P\ /\ s$ was identified at many places in the CSP literature [15, 27, 28], especially after basing the process-ordering ($\sqsubseteq$) on its refusals. *After* and $P\ /\ s$ play a pivotal role when linking CSP to automata-theoretic concepts; nevertheless it was commonly treated as something "meta"[5].

In our work, we turn *After* into an ordinary operator, compatible with all other concepts, preserving the invariant of $'\alpha\ process$, enjoying monotony and continuity, and a number of distributivities (that one could call "algebraic laws") useful for establishing an operational semantics. The formal proofs of this part of our theory amount to 2000 lines. For example, we obtain equalities like the following:

▶ **Theorem 6** ($After$ and $Sync$)**.** *$(P\ [\![S]\!]\ Q)$ after e is equal to:*

$$\begin{array}{l}\textsf{if } P = \bot \vee Q = \bot \textsf{ then } \bot\\ \textsf{else if } ev\ e \in P^0 \cap Q^0\\ \quad \textsf{then if } e \in S \textsf{ then } P \textit{ after } e\ [\![S]\!]\ Q \textit{ after } e\\ \qquad \textsf{else } (P \textit{ after } e\ [\![S]\!]\ Q) \sqcap (P\ [\![S]\!]\ Q \textit{ after } e)\\ \quad \textsf{else if } ev\ e \in P^0 \wedge e \notin S \textsf{ then } P \textit{ after } e\ [\![S]\!]\ Q\\ \qquad \textsf{else if } ev\ e \in Q^0 \wedge e \notin S \textsf{ then } P\ [\![S]\!]\ Q \textit{ after } e \textsf{ else } \Psi\ (P\ [\![S]\!]\ Q)\ e\end{array}$$

The only operator for which we have not managed to establish such a property is *Hiding*. However, we have at least the following monotonies:

▶ **Theorem 7** ($After$ and $Hiding$)**.**

$$\begin{array}{l}[\![ev\ e \in P^0;\ e \in B]\!] \Longrightarrow (P \setminus B) \sqsubseteq_{FD} (P \textit{ after } e \setminus B)\\ [\![ev\ e \in P^0;\ e \notin B]\!] \Longrightarrow (P \setminus B) \textit{ after } e \sqsubseteq_{FD} (P \textit{ after } e \setminus B)\end{array}$$

This will not be too restrictive for our construction thanks to the following theorem.

▶ **Theorem 8** (Characterization of FD-refinement)**.**
The FD-refinement $P \sqsubseteq_{FD} Q$ holds if and only if $P = P \sqcap Q$.

3.3 The Rationale for an Operational Semantics

With respect to the τ transition, ($\leadsto_\tau$), we follow the choice of Jifeng and Hoare in [20], i. e. we define it by $P \leadsto_\tau Q \equiv P \sqsubseteq_{FD} Q$.

With respect to the external transitions, we expect that:

- $P \leadsto_e Q$ (resp. $P \leadsto_\checkmark Q$) is impossible if $ev\ e \notin P^0$ (resp. $\checkmark \notin P^0$)
- event transitions should absorb τ transitions (on both sides) because ($\sqsubseteq_{FD}$) is transitive
- since $P \sqsubseteq_{FD} Q$ can be interpreted as "Q is more deterministic than P", Q should be at least as deterministic as P *after* e when P makes a transition via event e.

The formalization still requires an extension of the *After* operator to deal with ✓. This is achieved by extending the **locale** with an additional parameter Ω.

[5] Roscoe states that "this operator should not be thought of as an ordinary part of the CSP language"[27].

▶ **Definition 3** ($After_{tick}$ operator).

definition $After_{tick}$ :: ‹['α process, 'α event] ⇒ 'α process› (**infixl** ‹after✓› 77)
where ‹P after✓ e ≡ case e of ev x ⇒ P after x | ✓ ⇒ Ω P›

3.4 Finally: Formal Definitions of the Transition Relations

To make our construction as general as possible, we formalize the requirements of Subsection 3.3 by parameterizing the τ transition in a **locale** with four hypotheses:

locale *OpSemTransitions* = *AfterExt* Ψ Ω
for Ψ :: ‹['α process, 'α] ⇒ 'α process› **and** Ω :: ‹'α process ⇒ 'α process› +
fixes τ-trans :: ‹['α process, 'α process] ⇒ bool› (**infixl** ‹↝τ› 50)
assumes τ-trans-NdetL: ‹P ⊓ Q ↝τ P›
and τ-trans-transitivity: ‹P ↝τ Q ⟹ Q ↝τ R ⟹ P ↝τ R›
and τ-trans-anti-mono-initials: ‹P ↝τ Q ⟹ Q⁰ ⊆ P⁰›
and τ-trans-mono-AfterExt: ‹Q⁰ ⟹ P ↝τ Q ⟹ P after✓ e ↝τ Q after✓ e›
begin

abbreviation ev-trans :: ‹['α process, 'α, 'α process] ⇒ bool› (‹- ↝_ -› [50, 3, 51] 50)
where ‹P ↝e Q ≡ ev e ∈ P⁰ ∧ P after✓ ev e ↝τ Q›

abbreviation tick-trans :: ‹['α process, 'α process] ⇒ bool› (‹- ↝✓ -› [50, 51] 50)
where ‹P ↝✓ Q ≡ ✓ ∈ P⁰ ∧ P after✓ ✓ ↝τ Q›

To sum up, this **locale** needs to be instantiated with:

- a function Ψ that is a placeholder for the value of P *after* e when $ev\ e \notin P^0$
- a function Ω that is a placeholder for the value of $P\ after_{\checkmark}\ \checkmark$
- a binary relation $(\rightsquigarrow_\tau)$ which is compatible with $(\sqcap)$, is transitive, makes *initials* anti-monotonic and makes $After_{tick}$ monotonic.

With these only four local axioms, we can derive most of the basic operational rules for *SKIP*, $e \rightarrow P$, etc., and some of the rules for $P \setminus S$ or $P\ ;\ Q$, ... In order to recover the remaining rules, we divide the work. For each operator with a missing rule, we introduce a **locale** inheriting from *OpSemTransitions* in which we add a local axiom (about $(\rightsquigarrow_\tau)$). With the rules already proven and the denotational properties of the operator, we can derive the missing rules. Here, we illustrate the case of one of the rules for the *Sync* operator:

$$\frac{e \notin S \qquad P \rightsquigarrow_e P'}{P \ [\![S]\!]\ Q \rightsquigarrow_e P' \ [\![S]\!]\ Q}$$

▶ **Proof 1** (Derivation of one of the *Sync* Operational Rules).
Case $P = \bot$ or $Q = \bot$, this is obvious because of the properties of $\bot$.
Otherwise since $P \rightsquigarrow_e P'$ we have $ev\ e \in P^0$.
With Theorem 4 and $e \notin S$, we additionally have $ev\ e \in (P \ [\![S]\!]\ Q)^0$.
Thus, from Theorem 6, $(P \ [\![S]\!]\ Q)$ after e is equal to $(P$ after $e \ [\![S]\!]\ Q) \sqcap (P \ [\![S]\!]\ Q$ after $e)$ if $ev\ e \in Q^0$, and P after $e \ [\![S]\!]\ Q$ otherwise.
In both cases, with $(\rightsquigarrow_\tau)$ properties, we obtain $(P \ [\![S]\!]\ Q)$ after $e \rightsquigarrow_\tau P$ after $e \ [\![S]\!]\ Q$.
Using the additional assumption that, in general, $P \rightsquigarrow_\tau P' \Longrightarrow P \ [\![S]\!]\ Q \rightsquigarrow_\tau P' \ [\![S]\!]\ Q$, we finally conclude that $P \ [\![S]\!]\ Q \rightsquigarrow_e P' \ [\![S]\!]\ Q$.

Finally, by assembling the **locale**s of each operator (which is inheriting of all these **locale**s), their instantiations lead to formal proofs that the core of the ruleset shown in Section 4 are actually derivable for T,F, and FD semantics. In the end, they all rely on the four assumptions of *OpSemTransitions* and on eight additional assumptions of monotony for the first argument wrt. $(\leadsto_\tau)$ for the operators *Det*, *Seq*, *Hiding*, *Sync*, *Sliding*, *Interrupt*, *Throw* and *Renaming* e. g. $P \leadsto_\tau P' \Longrightarrow P \rhd Q \leadsto_\tau P' \rhd Q$. Special cases of rules not in the core ruleset will be discussed in the subsequent sections.

4 The Derived Rules of the Operational Semantics at a Glance

$$\frac{P \leadsto_e P' \qquad P' \leadsto_\tau P''}{P \leadsto_e P''} \qquad \frac{P \leadsto_\tau P' \qquad P' \leadsto_e P''}{P \leadsto_e P''}$$

$$\frac{P \leadsto_\checkmark P' \qquad P' \leadsto_\tau P''}{P \leadsto_\checkmark P''} \qquad \frac{P \leadsto_\tau P' \qquad P' \leadsto_\checkmark P''}{P \leadsto_\checkmark P''}$$

ABSORPTION

$$\frac{}{SKIP \leadsto_\checkmark \Omega\ SKIP}\ SKIP$$

$$\frac{cont\ f \qquad P = (\mu\ x.\ f\ x)}{P \leadsto_\tau f\ P}\ \text{FIXED POINT}$$

$$\frac{}{e \rightarrow P \leadsto_e P} \qquad \frac{e \in A}{\Box a{\in}A \rightarrow P\ a \leadsto_e P\ e} \qquad \frac{e \in A}{\sqcap a{\in}A \rightarrow\ P\ a \leadsto_e P\ e}$$

PREFIX

$$\frac{}{P \sqcap Q \leadsto_\tau P} \qquad \frac{}{P \sqcap Q \leadsto_\tau Q} \qquad \frac{e \in A}{\sqcap\ a{\in}A.\ P\ a \leadsto_\tau P\ e}$$

INTERNAL CHOICE

$$\frac{P \leadsto_\tau P'}{P \,\Box\, Q \leadsto_\tau P' \,\Box\, Q} \qquad \frac{P \leadsto_e P'}{P \,\Box\, Q \leadsto_e P'} \qquad \frac{P \leadsto_\checkmark P'}{P \,\Box\, Q \leadsto_\checkmark \Omega\ SKIP}$$

$$\frac{Q \leadsto_\tau Q'}{P \,\Box\, Q \leadsto_\tau P \,\Box\, Q'} \qquad \frac{Q \leadsto_e Q'}{P \,\Box\, Q \leadsto_e Q'} \qquad \frac{Q \leadsto_\checkmark Q'}{P \,\Box\, Q \leadsto_\checkmark \Omega\ SKIP}$$

EXTERNAL CHOICE

$$\frac{}{P \rhd Q \leadsto_\tau Q} \qquad \frac{P \leadsto_\tau P'}{P \rhd Q \leadsto_\tau P' \rhd Q} \qquad \frac{P \leadsto_e P'}{P \rhd Q \leadsto_e P'} \qquad \frac{P \leadsto_\checkmark P'}{P \rhd Q \leadsto_\checkmark \Omega\ SKIP}$$

SLIDING CHOICE

$$\frac{P \leadsto_\tau P'}{P\ ;\ Q \leadsto_\tau P'\ ;\ Q} \qquad \frac{P \leadsto_e P'}{P\ ;\ Q \leadsto_e P'\ ;\ Q} \qquad \frac{P \leadsto_\checkmark P' \qquad Q \leadsto_\tau Q'}{P\ ;\ Q \leadsto_\tau Q'}$$

SEQUENTIAL COMPOSITION

$$\frac{P \leadsto_\tau P'}{P \setminus B \leadsto_\tau P' \setminus B} \qquad \frac{P \leadsto_\checkmark P'}{P \setminus B \leadsto_\checkmark \Omega\ SKIP}$$

$$\frac{e \notin B \qquad P \leadsto_e P'}{P \setminus B \leadsto_e P' \setminus B} \qquad \frac{e \in B \qquad P \leadsto_e P'}{P \setminus B \leadsto_\tau P' \setminus B}$$

HIDING

$$\frac{P \leadsto_\tau P'}{P\ [\![S]\!]\ Q \leadsto_\tau P'\ [\![S]\!]\ Q} \qquad \frac{e \notin S \qquad P \leadsto_e P'}{P\ [\![S]\!]\ Q \leadsto_e P'\ [\![S]\!]\ Q} \qquad \frac{P \leadsto_\checkmark P'}{P\ [\![S]\!]\ Q \leadsto_\tau SKIP\ [\![S]\!]\ Q}$$

$$\frac{Q \leadsto_\tau Q'}{P\ [\![S]\!]\ Q \leadsto_\tau P\ [\![S]\!]\ Q'} \qquad \frac{e \notin S \qquad Q \leadsto_e Q'}{P\ [\![S]\!]\ Q \leadsto_e P\ [\![S]\!]\ Q'} \qquad \frac{Q \leadsto_\checkmark Q'}{P\ [\![S]\!]\ Q \leadsto_\tau P\ [\![S]\!]\ SKIP}$$

$$\frac{e \in S \qquad P \leadsto_e P' \qquad Q \leadsto_e Q'}{P\ [\![S]\!]\ Q \leadsto_e P'\ [\![S]\!]\ Q'} \qquad \frac{}{SKIP\ [\![S]\!]\ SKIP \leadsto_\checkmark \Omega\ SKIP}$$

SYNCHRONIZATION

$$\frac{P \leadsto_\tau P'}{P \triangle Q \leadsto_\tau P' \triangle Q} \qquad \frac{P \leadsto_e P'}{P \triangle Q \leadsto_e P' \triangle Q} \qquad \frac{P \leadsto_\checkmark P'}{P \triangle Q \leadsto_\checkmark \Omega\ SKIP}$$

$$\frac{Q \leadsto_\tau Q'}{P \triangle Q \leadsto_\tau P \triangle Q'} \qquad \frac{Q \leadsto_e Q'}{P \triangle Q \leadsto_e Q'} \qquad \frac{Q \leadsto_\checkmark Q'}{P \triangle Q \leadsto_\checkmark \Omega\ SKIP}$$

INTERRUPT

$$\frac{P \leadsto_\tau P'}{P\ \Theta\ a{\in}A.\ Q\ a \leadsto_\tau P'\ \Theta\ a{\in}A.\ Q\ a} \qquad \frac{P \leadsto_\checkmark P'}{P\ \Theta\ a{\in}A.\ Q\ a \leadsto_\checkmark \Omega\ SKIP}$$

$$\frac{e \notin A \qquad P \leadsto_e P'}{P\ \Theta\ a{\in}A.\ Q\ a \leadsto_e P'\ \Theta\ a{\in}A.\ Q\ a} \qquad \frac{e \in A \qquad P \leadsto_e P'}{P\ \Theta\ a{\in}A.\ Q\ a \leadsto_e Q\ e}$$

THROW

$$\frac{P\ {}_\alpha\!\!\leadsto_\tau P'}{Renaming\ P\ f\ {}_\beta\!\!\leadsto_\tau Renaming\ P'\ f}$$

$$\frac{f\ a = b \qquad P\ {}_\alpha\!\!\leadsto_a P'}{Renaming\ P\ f\ {}_\beta\!\!\leadsto_b Renaming\ P'\ f} \qquad \frac{P\ {}_\alpha\!\!\leadsto_\checkmark P'}{Renaming\ P\ f\ {}_\beta\!\!\leadsto_\checkmark \Omega_\beta\ SKIP}$$

RENAMING

5 Big Steps Semantics

The notation $P \ / \ [e]$ sometimes appearing in the classical literature will now be given a formal definition in terms of the $After_{tick}$ (Definition 3) operator. From there, the generalization to an operator "connecting" two processes P and Q via a trace s — analogously to the notion of δ function in automata theory — is straightforward. In the same manner, we can combine small steps transitions to a trace transition. These generalized notions will allow for both establishing new formats of refinement proofs as well as (bi)simulation theorems.

5.1 Extensions to Traces

The definition of the generalized *After* operator proceeds inductively:

▶ **Definition 4** ($After_{trace}$ operator).

fun $After_{trace} :: \langle['\alpha\ process, '\alpha\ trace] \Rightarrow '\alpha\ process\rangle$ (**infixl** $\langle after_{\mathcal{T}}\rangle$ 77)
where $\langle P\ after_{\mathcal{T}}\ [] = P\rangle$
| $\langle P\ after_{\mathcal{T}}\ (e\ \#\ s) = (P\ after_{\checkmark}\ e)\ after_{\mathcal{T}}\ s\rangle$

The definition of the generalized trace-transition is done analogously:

▶ **Definition 5** (Transition with a trace).

inductive *trace-trans* $:: \langle['\alpha\ process, '\alpha\ trace, '\alpha\ process] \Rightarrow bool\rangle$ ($\langle$-/ $\leadsto^*$-/ -$\rangle$ [50, 3, 51] 50)
where *trace-τ-trans* : $\langle P \leadsto_\tau P' \Longrightarrow P \leadsto^* [] P'\rangle$
| *trace-tick-trans* : $\langle P \leadsto_{\checkmark} P' \Longrightarrow P \leadsto^* [\checkmark] P'\rangle$
| *trace-Cons-ev-trans* : $\langle P \leadsto_e P' \Longrightarrow P' \leadsto^* s\ P'' \Longrightarrow P \leadsto^* (ev\ e)\ \#\ s\ P''\rangle$

The $After_{trace}$ operator and the trace transition $P \leadsto^* s\ Q$ are deeply related, which is expressed in the following theorem:

▶ **Theorem 9** (Bridge between trace transition and $After_{trace}$ operator).

$P \leadsto^* s\ Q$ if and only if $s \in \mathcal{T}\ P \wedge P\ after_{\mathcal{T}}\ s \leadsto_\tau Q$.

Informally spoken, $P\ after_{\mathcal{T}}\ s$ is the the least deterministic process that we can expect from process P after the trace s. This interpretation will become clearer when we will instantiate the formal $(\leadsto_\tau)$-relation of the locale with concrete refinements in Section 6.

Since the projections for the $After_{trace}$ operator are relatively easy to handle, this theorem is an important new weapon in our arsenal. We will illustrate this by the following "reality checks" which have concrete applications when certifying traces or divergences.

▶ **Theorem 10** (Reality Checks).

- *We have $s \in \mathcal{T}\ P$ if and only if $\exists\ Q.\ P \leadsto^* s\ Q$.*
- *Under the assumption $\forall\ P.\ P \leadsto_\tau \bot \longrightarrow P = \bot$ of unicity of the least element of $(\leadsto_\tau)$, a trace tickFree s verifies $s \in \mathcal{D}\ P$ if and only if $P \leadsto^* s\ \bot$.*
- *Under the assumption $\forall\ P\ Q.\ P \leadsto_\tau Q \longrightarrow P \sqsubseteq_F Q$ that a τ transition implies F-refinement, and if tickFree s, we have $(s, X) \in \mathcal{F}\ P$ if and only if $\exists\ Q.\ P \leadsto^* s\ Q \wedge X \in \mathcal{R}\ Q$.*

(where $\mathcal{R}\ Q$ is the set of refusals of Q, defined as $\{X \mid ([], X) \in \mathcal{F}\ Q\}$).

5.2 Strong Induction and (Bi)Simulations

The following theorem (and its generalizations not shown here) represents a new form of induction over the set of reachable processes:

▶ **Theorem 11** (Strong Induction for Refinements). *Let f be a continuous function:*

$$\begin{array}{l}[\![\exists\, s{\in}\mathcal{T}\ P.\ tickFree\ s \wedge Q = P\ after_{\mathcal{T}}\ s;\\ \quad \bigwedge s\ x.\ [\![s \in \mathcal{T}\ P;\ \forall\, y.\ (\exists\, s{\in}\mathcal{T}\ P.\ tickFree\ s \wedge y = P\ after_{\mathcal{T}}\ s) \longrightarrow x \sqsubseteq_{FD} y]\!]\\ \qquad \Longrightarrow f\ x \sqsubseteq_{FD} P\ after_{\mathcal{T}}\ s]\!]\\ \Longrightarrow (\mu\ x.\ f\ x) \sqsubseteq_{FD} Q\end{array}$$

Note that as in Hoare and Jifengs approach, there will always be infinite sequences of τ transitions. This is a consequence of the fact that the FD-refinement is reflexive. More generally speaking, since any process equality $P = Q$ is subsumed by the reflexivity $P \rightsquigarrow_\tau Q$, this is unavoidable: unfolding fixed points, for example, also falls into the category of infinite sequences of τ transitions. Compared to classical operational semantics in CCS, which is defined purely in terms of syntactic manipulations on process-terms, this means that we can never have "strong simulations" in our denotational framework, which is based on higher-order abstract syntax and a congruence generated from the equality on the semantic domain. Rather, we will target *weak* transitions resp. simulations, which are, as we argue, more suited for the semantic treatment we are heading for.

6 The Construction put into a Global Perspective

6.1 Transitions as Local Refinements

We use a **sublocale** to partially instantiate *OpSemTransitions* with $(\sqsubseteq_{FD})$, i. e. by leaving Ψ and Ω as parameters. In this context we prove that we have only one remaining hypothesis : $[\![\checkmark \in Q^0;\ P\ {}_{FD}\!\rightsquigarrow_\tau Q]\!] \Longrightarrow \Omega\ P\ {}_{FD}\!\rightsquigarrow_\tau \Omega\ Q$, which is obvious if Ω takes the value $STOP$ (a choice commonly used in the literature, whether explicitly stated or implied). However, in principle, any constant function is acceptable for Ω since we do not care about the traces after a termination[6].

Independent of this choice, we recover all the rules of operational semantics, and even better for the "reality checks" (Theorem 10) since we get rid of the hypotheses.

However, it remains unfortunate that in the right hand side of the equivalence for failures appears the denotational notion of refusals.

One work-around for this problem are *deterministic processes*. A process P is said to be *deterministic* if $\forall\, s\ e.\ s\ @\ [e] \in \mathcal{T}\ P \longrightarrow (s, \{e\}) \notin \mathcal{F}\ P$. We will not detail much this notion here[7], but to cut a long story short we prove that being in the refusals set of a *deterministic* process P is the same as non intersecting its *initials* P^0. Moreover, the notion of *deterministic* is preserved by *tickFree* trace transitions. We finally prove:

▶ **Theorem 12** (Deterministic Version of failures reality Check).
Assuming deterministic P and tickFree s, we have:

$$(s,\ X) \in \mathcal{F}\ P \text{ if and only if } \exists\, Q.\ P\ {}_{FD}\!\rightsquigarrow^* s\ Q \wedge X \cap Q^0 = \emptyset.$$

It came as a pleasant surprise when we observed that the same argument applies for the trace divergence refinement $(\sqsubseteq_{DT})$. Initially defined and studied in [31] for pure curiosity, it

[6] Therefore, we are free to choose for Ω $\lambda P.\ P \sqcap STOP$ or $\lambda P.\ Renaming\ P\ f$ or $\lambda P.\ e \rightarrow P$, ...
[7] We refer to [27].

behaves remarkably well: the only remaining hypothesis is a monotony for $\Omega : [\![\checkmark \in Q^0;\ P\ {}_{DT}\!\!\rightsquigarrow_\tau\ Q]\!] \Longrightarrow \Omega\ P\ {}_{DT}\!\!\rightsquigarrow_\tau\ \Omega\ Q$ and we recover all the rules of the operational semantics.

Of course by restricting ourselves to traces and the divergences, we can not reason about failures anymore.

It turns out that it is also possible to instantiate the τ transition in the **locale** *OpSem-Transitions* with the failure refinement or the trace refinement. However, for them, the result is somewhat unimpressive. The following table summarizes the rule sets corresponding to operators which can be established.

	basic	$(\Box)$	$(;)$	$P \setminus S$	$P\ [\![S]\!]\ Q$	$(\triangleright)$	$(\triangle)$	*Throw*	*Renaming*
$(\sqsubseteq_{FD})$	✓	✓	✓	✓	✓	✓	✓	✓	✓
$(\sqsubseteq_{DT})$	✓	✓	✓	✓	✓	✓	✓	✓	✓
$(\sqsubseteq_{F})$	✓	✗	✗	✓	✗	✗	✗	✗	✗
$(\sqsubseteq_{T})$	✓	✓	✗	✓	✗	✓	✓	✗	✗

where "basic" refers to the rules for absorption, *SKIP*, $e \rightarrow P$, $\Box\ a \in A \rightarrow P\ a$, $\sqcap\ a \in A \rightarrow P\ a$, $P \sqcap Q$ and $\mu\ X.\ f\ X$ that we get as soon as we instantiate *OpSemTransitions*.

Being able to instantiate the locale represents the main result: we can formally derive an operational semantics from the denotational one developed in `HOL-CSP`. This also constitutes a formal proof that the expected rules are consistent (see Section 4).

And last but not least, we obtained actually two interesting variants with FD-refinement and DT-refinement, plus all the sub-variants resulting from the free choice of Ψ and Ω.

6.2 Running Example: the Copy Buffer Again

From the derived laws of Section 4 we can formally obtain the following LTSs for the Copy Buffer example presented in Subsection 2.5. Note that the τ-transition are collapsed thanks to the absorbtion rules, and with the properties of *initials* we ensure that no external transition is missed. Further note that *t1* is a key for the term $mid\ e \rightarrow ack \rightarrow SEND$, *t2* for $ack \rightarrow SEND$, *t3* for $right\ e \rightarrow ack \rightarrow REC$, *t4* for $ack \rightarrow REC$, and finally *t5* for $right\ e \rightarrow SYS$.

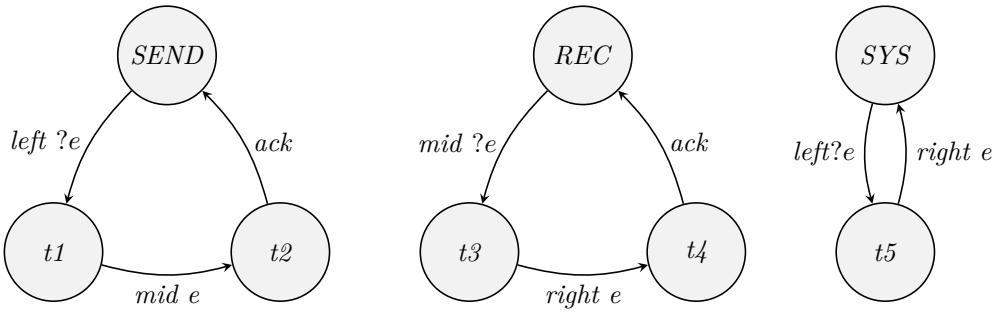

Figure 1 LTSs for the Copy Buffer Example

6.3 Comparison with the Work of Jifeng and Hoare

As mentioned in Section 3, Hoare and Jifeng in [20] defined $P \rightsquigarrow_\tau Q \equiv P \sqsubseteq_{FD} Q$ and

$P \leadsto_e Q \equiv P \sqsubseteq_{FD} (e \to Q) \,\square\, P$. Looking at our version (instantiated with the FD-refinement), we differ on the external transition with $ev\ e \in P^0 \wedge P\ after\ e \sqsubseteq_{FD} Q$ instead. What happened? Did we miss something?

In general, from the generic τ transition in our **locale**, we can define $P\ {}_{HOARE}\!\!\leadsto_e Q \equiv P \leadsto_\tau (e \to Q) \,\square\, P$. We immediately prove that their version is stronger than ours i. e. $P\ {}_{HOARE}\!\!\leadsto_e Q \Longrightarrow P \leadsto_e Q$. However, adding the two hypotheses of monotony on $(\leadsto_\tau)$, we can prove the reciprocal. The situation is summarized in the following theorem:

▶ **Theorem 13** (Equivalence of Transitions).
Assuming a τ monotony for prefix: $\forall P\ P'\ e.\ P \leadsto_\tau P' \longrightarrow e \to P \leadsto_\tau e \to P'$
and for Det: $\forall P\ P'\ Q.\ P = \bot \vee P' \neq \bot \longrightarrow P \leadsto_\tau P' \longrightarrow P \,\square\, Q \leadsto_\tau P' \,\square\, Q$,
we have $P\ {}_{HOARE}\!\!\leadsto_e Q$ if and only if $P \leadsto_e Q$.

These two hypotheses are verified by all four refinement relations. In other words, the definition of Jifeng and Hoare is equivalent to ours as long as we do not consider ✓!

Indeed, as mentioned in Section 3, their definition can not handle ✓ because the *prefix* operator only accepts a "real" event. In this sense one can say that our construction is a generalization. Furthermore, the $After_{tick}$ operator gives a direct access to the least deterministic process that can be expected while doing an external transition, which is not easily accessible from the version of Hoare and Jifeng. Finally we note that the *After* operator is itself of interest, even if we restrict ourselves to a purely denotational reasoning[8].

6.4 Discussion

Our construction and the resulting proof rules (Section 4) permit the following observations:

1. As a general rule, when looking at a transition involving the special event ✓, we obtain something like $P \otimes Q \leadsto_\checkmark \Omega\ SKIP$. This is a consequence of Theorem 5 and Theorem 8.
2. The "absorption" rules at the beginning allow additional rules to be derived directly e. g.
$$\frac{P \leadsto_\checkmark P' \qquad Q \leadsto_e Q'}{P\ ;\ Q \leadsto_e Q'} \qquad \frac{P \leadsto_\checkmark P' \qquad Q \leadsto_\checkmark Q'}{P\ ;\ Q \leadsto_\checkmark Q'}$$
3. About the termination of *Sync* operator, Roscoe postulates in [27] that:
$$\frac{P \leadsto_\checkmark P'}{P \ [\![S]\!]\ Q \leadsto_\tau \Omega' \ [\![S]\!]\ Q} \qquad \frac{Q \leadsto_\checkmark Q'}{P \ [\![S]\!]\ Q \leadsto_\tau P \ [\![S]\!]\ \Omega'} \qquad \frac{}{\Omega' \ [\![S]\!]\ \Omega' \leadsto_\checkmark \Omega'}$$
where Ω' is intended to denote any process that has already terminated. In the common interpretation that Ω' can be identified with *STOP*, these rules are incompatible with the denotational properties since we have $STOP \ [\![S]\!]\ STOP = STOP$ that can not make a ✓ transition. Under the assumptions of the **locale**, we rather prove the rules of Section 4.
4. We deliberately focus in Section 4 on the operational rules that we found in the literature. In particular for the *Throw* operator, where the right argument remains inactive until an exception is triggered, we should not write a right τ transition rule like:
$$\frac{\forall\, a{\in}A.\ Q\ a \leadsto_\tau Q'\ a}{P\ \Theta\ a{\in}A.\ Q\ a \sqsubseteq_{FD} P\ \Theta\ a{\in}A.\ Q'\ a}$$
while this is true when instantiating $(\leadsto_\tau)$ with $(\sqsubseteq_{FD})$, $(\sqsubseteq_{DT})$, $(\sqsubseteq_F)$ or $(\sqsubseteq_T)$.

[8] The interested reader is referred to examples of fixed point induction to reason about deadlock freeness[4].

7 Related Work

In the introduction, we claimed that HOL-CSP is arguably the most *comprehensive* formalization of CSP; here, we'd like to substantiate this claim.

The theory of CSP has attracted a lot of interest since the eighties and nineties, both as a theoretical device as well as a modelling language to analyze complex concurrent systems. A wealth of theoretical articles appeared to investigate certain fragments and extensions of the core framework; it is therefore not surprising that attempts to their formalisations have been undertaken with the advent of interactive proof assistants.

Most noteworthy to these attempts is an early CSP trace semantics model in HOL System proposed by [11]. Its successor [10] presented a first failure-divergence semantics for a restricted set of operators and used the notion of a universal (polymorphic) alphabet[9]. Note that [34] tackled already with subtle difficulties concerning *is-process* and ✓.

The tool CSP-Prover [18] — based on a deep embedding of CSP in an Isabelle/HOL theory on the stable failures model — allows for the refinement verification [18] by using some automated support for induction. However, only if a process is divergence-free, its failures are the same as its stable failures. In our view, this is a too strong assumption for both a theory as well as a practical tool.

In the past few years, CSP benefited from a renewed interest with proof assistants. CSP Agda was introduced in 2026 [16] with an implementation quite different from `HOL-CSP` since it is based on coinductive data types. Only trace and stable failures semantics have been covered so far, and the library of proven laws is fairly modest [17]. In 2020, an operational semantics of CSP in Coq was introduced [13] by a direct definitional approach. The theory covers only trace refinement and a subset of CSP's operators, but offers rather well-developed proof automation for this language fragment close to conventional automata theory.

With respect to all these formalizations in HOL, we would like to remind the importance of the general fact that invariants (like *is-process*) or bridge theorems (like Hoare's $P \rightsquigarrow_e Q \equiv P \sqsubseteq_{FD} (e \rightarrow Q) \,\Box\, P$) do not simply generalise from one fragment to the next, and that features which are well studied in one fragment are not necessarily well-understood in the whole picture. It is our main contribution to provide an integrated formal theory that tackles with the complexities of the necessary generalisations. This involved a revision of the role of the *After* operator in the entire theory.

In the late nineties, research focused on automated verification tools for CSP, most notably on FDR (see [1] for the latest instance). It relies on an operational CSP semantics, that allows for a conversion of processes into labelled transition systems, where the states were normalized by the "axioms" derived from the denotational semantics. For finite event sets, FDR can reduce refinement proofs to bisimulation problems. With efficient compression techniques, state-elimination and factorization by semantic equivalence [26], FDR was successful in analysing some industrial applications. However, such a model checker can never handle infinite cases. Another similar model checking tool [30] implemented some more optimization techniques, such as partial order reduction, symmetric reduction, and parallel model checking, but is also restricted to the finite case. In a way, these tool require for their foundation integrated denotational/algebraic/operational techniques as provided by our theory.

Attempts to find characterizations of processes to generalise finite results to infinite ones by *data-independence* [2, 21, 25], a variant of parametric model-checking, have seen only a

[9] Our first attempts for HOL-CSP [34] are based on a extended version of this theory ported to Isabelle/HOL

limited success. Roscoe developed a data independent technology to verify security protocols modelled with CSP/FDR, which allows the node to call infinite fresh values for nonces, thus infinite sequence of operations [25]. An extension of this work was proposed in [2] using the script language CSP_M. However, in their works, even though each agent in the security protocol can perform infinite number of operations, the number of agent entities remains finite. HOL-CSP satisfies the need to parameterization and high-order processes naturally [32] as a consequence of their *pcpo*-type structure. A formalization and theory development of CSP_M has been undertaken [3] but is out of the scope of this paper.

8 Conclusion

We presented a formalisation of a comprehensive semantic theory for CSP, a 'classical' language for the specification and analysis of concurrent systems studied in a rich body of literature. The theory comprises the denotational part (including recursion and permitting higher-order processes), the algebraic part paving the way for parametric refinement proofs involving fixed point induction, and the operational semantics part. The size of the latter, which constitutes an original contribution, is about 16 kLOC of Isabelle/HOL proofs.

The resulting framework offers new ways to reason consistently over denotationally defined CSP processes paving the way to symbolic execution of LTS-based representations of processes as well as the possibility to *certify* output from model-checkers like [1, 30], which excel in the calculational parts related to interleaving processes. As a by-product, the theory allows for new proof principles like strong induction (cf. Theorem 11).

An interesting line of future work is the development of a library of "process-bricks" containing, e. g., semaphores, monitors or just global variables like:

$$VAR\ Read\ Update \equiv \mu\ x.\ (\lambda\sigma.\ (Read!\sigma \rightarrow x\ \sigma) \ \square\ (Update?\sigma' \rightarrow x\ \sigma'))$$

or non-deterministic key-generators like:

$$KEY\ chan \equiv (\mu\ x.\ (\lambda\sigma.\ chan!a \in \sigma \rightarrow x\ (\sigma - \{a\}))) \ \mathbb{N}$$

which will have symbolic traces like:

$$[\![a \in \mathbb{N};\ b \in \mathbb{N} - \{a\};\ c \in \mathbb{N} - \{a, b\}]\!] \Longrightarrow [C\ a,\ C\ b,\ C\ c] \in \mathcal{T}\ (KEY\ C)$$

which can be derived both algebraically as well as operationally.

References

1 FDR4 - The CSP Refinement Checker. https://www.cs.ox.ac.uk/projects/fdr/, 2019.

2 Jing An, Lei Zhang, and Chun You. The design and implementation of data independence in the csp model of security protocol. *Advanced Materials Research*, 915-916:1386–1392, 04 2014. doi:10.4028/www.scientific.net/AMR.915-916.1386.

3 Benoît Ballenghien, Safouan Taha, and Burkhart Wolff. HOL-CSPM - Architectural operators for HOL-CSP. *Archive of Formal Proofs*, 2023, 2023. URL: https://www.isa-afp.org/entries/HOL-CSPM.html.

4 Benoît Ballenghien and Burkhart Wolff. Operational Semantics formally proven in HOL-CSP. *Archive of Formal Proofs*, December 2023. URL: https://isa-afp.org/entries/HOL-CSP_OpSem.html.

5 G. Barrett. Model checking in practice: the t9000 virtual channel processor. *IEEE Transactions on Software Engineering*, 21(2):69–78, Feb 1995. doi:10.1109/32.345823.

6 S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

7 S. D. Brookes and A. W. Roscoe. An improved failures model for communicating sequential processes. In Stephen D. Brookes, Andrew William Roscoe, and Glynn Winskel, editors, *Seminar on Concurrency*, pages 281–305, Berlin, Heidelberg, 1985. Springer.

8 Achim D. Brucker, Idir Aït-Sadoune, Nicolas Méric, and Burkhart Wolff. Using deep ontologies in formal software engineering. In Uwe Glässer, José Creissac Campos, Dominique Méry, and Philippe A. Palanque, editors, *Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings*, volume 14010 of *Lecture Notes in Computer Science*, pages 15–32. Springer, 2023. doi:10.1007/978-3-031-33163-3_2.

9 Achim D. Brucker and Burkhart Wolff. Isabelle/dof, July 2022. doi:10.5281/zenodo.6810799.

10 Albert J. Camilleri. A higher order logic mechanization of the csp failure-divergence semantics. In Graham Birtwistle, editor, *IV Higher Order Workshop, Banff 1990*, pages 123–150, London, 1991. Springer.

11 Albert John Camilleri. Mechanizing CSP trace theory in higher order logic. *IEEE Trans. Software Eng.*, 16(9):993–1004, 1990.

12 Paolo Crisafulli, Safouan Taha, and Burkhart Wolff. Modeling and analysing cyber-physical systems in HOL-CSP. *Robotics Auton. Syst.*, 170:104549, 2023. URL: https://doi.org/10.1016/j.robot.2023.104549, doi:10.1016/J.ROBOT.2023.104549.

13 Carlos Alberto da Silva Carvalho de Freitas. A theory for communicating, sequential processes in coq. 2020. URL: https://api.semanticscholar.org/CorpusID:259373665.

14 A.A.A. Donovan and B.W. Kernighan. *The Go Programming Language*. Addison-Wesley Professional Computing Series. Pearson Education, 2015.

15 C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

16 Bashar Igried and Anton Setzer. Programming with monadic csp-style processes in dependent type theory. In *Proceedings of the 1st International Workshop on Type-Driven Development*, TyDe 2016, page 28–38, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2976022.2976032.

17 Bashar Igried and Anton Setzer. Trace and stable failures semantics for csp-agda. *arXiv preprint arXiv:1709.04714*, 2017.

18 Yoshinao Isobe and Markus Roggenbach. A complete axiomatic semantics for the CSP stable-failures model. In *CONCUR 2006 - Concurrency Theory, 17th International Conference, Bonn, Germany, August 27-30, 2006*, pages 158–172, 2006.

19 Yoshinao Isobe and Markus Roggenbach. Csp-prover: a proof tool for the verification of scalable concurrent systems. *Information and Media Technologies*, 5(1):32–39, 2010. doi:10.11185/imt.5.32.

20 He Jifeng and CAR Hoare. From algebra to operational semantics. *Information Processing Letters*, 45(2):75–80, 1993.

21 Ranko S. Lazic. *A semantic study of data-independence with applications to the mechanical verification of concurren*. PhD thesis, University of Oxford, 1999.

22 Olaf Müller, Tobias Nipkow, David von Oheimb, and Oskar Slotosch. HOLCF = HOL + LCF. *j-fp*, 9(2):191–223, 1999. doi:10.1017/S095679689900341X.

23 Pasquale Noce. Conservation of CSP noninterference security under sequential composition. *Archive of Formal Proofs*, 2016. URL: https://www.isa-afp.org/entries/Noninterference_Sequential_Composition.shtml.

24 A. W. Roscoe. An alternative order for the failures model. *J. Log. Comput.*, 2:557–577, 1992.

25 A. W. Roscoe and Philippa J. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7(1):147–190, 1999.

26 A. W. Roscoe, P. H. B. Gardiner, M. H. Goldsmith, J. R. Hulance, D. M. Jackson, and J. B. Scattergood. Hierarchical compression for model-checking csp or how to check 1020 dining philosophers for deadlock. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 133–152, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

27 A.W. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall, 1997.

28 A.W. Roscoe. *Understanding Concurrent Systems*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

29 Dana Scott. Continuous lattices. In F. W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, pages 97–136, Berlin, Heidelberg, 1972. Springer.

30 Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. Pat: Towards flexible verification under fairness. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, pages 709–714, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

31 Safouan Taha, Burkhart Wolff, and Lina Ye. The HOL-CSP refinement toolkit. *Arch. Formal Proofs*, 2020, 2020. URL: `https://www.isa-afp.org/entries/CSP_RefTK.html`.

32 Safouan Taha, Burkhart Wolff, and Lina Ye. Philosophers may dine - definitively! In Brijesh Dongol and Elena Troubitsyna, editors, *Integrated Formal Methods - 16th International Conference, IFM 2020, Lugano, Switzerland, November 16-20, 2020, Proceedings*, volume 12546 of *Lecture Notes in Computer Science*, pages 419–439. Springer, 2020. `doi:10.1007/978-3-030-63461-2\_23`.

33 Safouan Taha, Lina Ye, and Burkhart Wolff. HOL-CSP Version 2.0. *Archive of Formal Proofs*, April 2019. URL: `http://isa-afp.org/entries/HOL-CSP.html`.

34 Haykal Tej and Burkhart Wolff. A corrected failure divergence model for CSP in Isabelle/HOL. In John S. Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *Formal Methods Europe (FME)*, volume 1313 of *LNCS*, pages 318–337. Springer, 1997. `doi:10.1007/3-540-63533-5_17`.

35 Jim Woodcock and Ana Cavalcanti. The semantics of circus. In Didier Bert, Jonathan P. Bowen, Martin C. Henson, and Ken Robinson, editors, *ZB 2002: Formal Specification and Development in Z and B, 2nd International Conference of B and Z Users, Grenoble, France, January 23-25, 2002, Proceedings*, volume 2272 of *Lecture Notes in Computer Science*, pages 184–203. Springer, 2002. `doi:10.1007/3-540-45648-1\_10`.

36 Jim Woodcock, Ana Cavalcanti, Simon Foster, Marcel Oliveira, Augusto Sampaio, and Frank Zeyda. Utp, circus, and isabelle. In Jonathan P. Bowen, Qin Li, and Qiwen Xu, editors, *Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 80th Birthday*, volume 14080 of *Lecture Notes in Computer Science*, pages 19–51. Springer, 2023. `doi:10.1007/978-3-031-40436-8\_2`.